1 *[5 puntos] Sea la siguiente implementación de una barrera de sincronización:*

```
struct bar_type {
    int counter = 0;
    struct lock_type lock;
    int flag;
} ;

void barrier (bar_type barrier_name, int n_procs)
{
    int mycount;

    lock (barrier_name.lock);
    if (barrier_name.counter == 0)
        barrier_name.flag = 0;
    mycount = barrier_name.counter++;
    unlock (barrier_name.lock);
    if (mycount == n_procs) {
        barrier_name.counter = 0;
        barrier_name.flag = 1;
    } else
        while (barrier_name.flag == 0) {};
}
```

a) *En general, en cuanto a las implementaciones de las barreras de sincronización, indique las ventajas e inconvenientes de realizarlas mediante hardware o mediante software. Indique también cuáles son más utilizadas.*

b) *Indique por qué no funciona la implementación de este enunciado y cuál es el problema que presenta.*

c) *Realice los cambios que considere oportunos en la implementación anterior con el objetivo de corregir el problema detectado. Puede detallarlos en esta misma hoja, a la derecha del código del enunciado, o bien describirlos en una hoja aparte.*

SOLUCIÓN

a) Las barreras de sincronización implementadas por hardware son algo más rápidas, pero necesitan líneas adicionales en el bus y son muy difíciles de implementar cuando no implican a todos los procesadores o cuando el número de procesos y sus identidades van variando de una llamada a otra de la barrera. En la vida real las implementaciones suelen ser todas software, por su facilidad y flexibilidad, reduciendo la posible contención a través de implementaciones distribuidas.

b) No funciona porque puede producirse un interbloqueo. Si alguno de los procesos que está esperando en el `while` a que cambie el flag pierde la CPU por un tiempo, podría ocurrir que el último proceso que llegue a la barrera cambie el flag (poniéndolo a 1) de modo que varios procesos abandonen la barrera y alguno de ellos vuelva a llamar a la misma barrera, cambiando de nuevo el flag a 0 antes de que un proceso que perdió el uso de CPU con el flag todavía a 0 haya podido recuperar la CPU. Cuando por fin recupere el control, seguirá viendo el flag a 0 y nunca saldrá de la barrera, mientras todos los demás estarán esperando en la próxima barrera.

c) Una solución consiste en cambiar el significado del flag de la barrera, de modo que en lugar de indicar con un "1" que se puede continuar la ejecución y con un "0" que se debe esperar por otros procesos, en cada nueva llamada se alterne el significado del flag, es decir, se intercambie el significado de los valores "0" y "1". Para ello se hace uso de una variable local, en este ejemplo, `local_sense`:

```
void barrier (bar_type barrier_name, int n_procs)
{
    local_sense = !(local_sense);
    lock (barrier_name.lock);
    barrier_name.counter++;
    if (barrier_name.counter == n_procs) {
        unlock (barrier_name.lock);
        barrier_name.counter = 0;
        barrier_name.flag = local_sense;
    } else {
        unlock (barrier_name.lock);
        while (barrier_name.flag != local_sense) {};
    }
}
```

2 *[5 puntos] A continuación se presenta un fragmento de código, escrito para un computador secuencial, que cuenta el número de elementos negativos, nulos y positivos de un vector de N elementos, donde N es un valor positivo.*

```
(1)        add  r1,  r0,  r0
(2)        add  r2,  r0,  r0
(3)        add  r3,  r0,  r0
(4)bucle:  ld   r13, r10, r0      ; Lee v[i]
(5)        blt  r13, r0,  $menor  ; si < 0
(6)        bgt  r13, r0,  $mayor  ; si > 0
(7)igual:  add  r2,  r2,  1       ; cont-ceros++
(8)        br   $fin
(9)menor:  add  r1,  r1,  1       ; cont-neg++
(10)       br   $fin
(11)mayor: add  r3,  r3,  1       ; cont-pos++
(12)fin:   add  r10, r10, 4
(13)       sub  r11, r11, 1
(14)       bne  r11, r0,  $bucle
(15)       st   r1,  r12, 0
(16)       st   r2,  r12, 4
(17)       st   r3,  r12, 8
```

```
cont-neg = 0;
cont-ceros = 0;
cont-pos = 0;
do {
    aux= v[i];
    if (aux < 0)
        cont-neg++;
    else if (aux > 0)
        cont-pos++;
    else cont-ceros++;
    i--;
} while (i >0)
res[0] = cont-neg;
res[1] = cont-ceros;
res[2] = cont-pos;
```

Se quiere ejecutar este código en un computador cuyo procesador tiene un pipeline de instrucciones con las 5 etapas que se indican a continuación (entre paréntesis se muestra el tiempo real que tardan en realizarse las operaciones de cada una de ellas). Todas las instrucciones pasan por todas las etapas, pudiendo no realizarse en alguna de ellas trabajo útil.

- *E1: Fetch (2ns)*
- *E2: Decodificación, lectura de registros y cálculo de todo tipo de direcciones (2ns)*
- *E3: Ejecución, evaluación de la condición en los saltos condicionales y carga del registro de datos en las instrucciones* `st` *(1ns)*
- *E4: Acceso a memoria para datos (2ns)*
- *E5: Escritura en registros (1ns)*

Este procesador resuelve las dependencias de control introduciendo ciclos de espera.

a) *Indique razonadamente cuál es el tiempo de ciclo del pipeline de este procesador.*

b) *Indique razonadamente las dependencias que generan parones en este código (ya sean de datos o de control), así como el número de ciclos de espera que debe introducir el procesador para resolverlas en los siguientes casos:*

b.1) *El procesador no dispone de mecanismos de adelantamiento.*

b.2) *El procesador dispone de todo tipo de mecanismos de adelantamiento.*

c) *Calcule el CPI de este fragmento de código en este procesador. Para ello suponga que dispone de todo tipo de adelantamientos y que el 25 % de los elementos son nulos y el 35 % son negativos.*

d) *Reordene el código para minimizar los ciclos de espera y calcule la ganancia obtenida con respecto al apartado anterior. Indique si considera significativa dicha ganancia y por qué.*

e) *Suponga ahora que se sustituye el procesador por uno idéntico, excepto en que tiene predicción dinámica de 1 bit, y que ésta se realiza en la etapa de Fetch. Calcule la tasa de fallos de predicción en la ejecución de la instrucción (14) del código original para $N = 10.000$, así como la reducción en el número de ciclos debidos a ejecución de dicha instrucción con respecto al procesador original. Para ello, suponga que la primera vez que se ejecuta se predice que no se salta.*

SOLUCIÓN

a) El tiempo de ciclo del pipeline es 2ns puesto que viene determinado por la duración de la etapa más lenta

b) Las dependencias que generan parones en el pipeline son las siguientes:

- Dependencias de datos:
  - Entre las instrucciones 4 y 5 por el registro r13.

- Entre las instrucciones 13 y 14 por el registro r11.

Sin adelantamiento, el procesador debe introducir en ambos casos 3 ciclos de espera, ya que hasta que no finaliza la etapa E5 de la instrucción que genera la dependencia (la 4 o la 13), la instrucción afectada (la 5 o la 14) no puede leer los valores correctos de los registros, en la etapa E2.

Si se utiliza adelantamiento, para resolver la primera dependencia y utilizando adelantamiento de la salida de la memoria a la entrada de la evaluación de la condición (E4 $->$ E3) se debe introducir solo 1 ciclo de espera. La segunda dependencia se puede resolver sin ningún ciclo de espera, utilizando adelantamiento de la salida de la ALU a la entrada de la evaluación de la condición, ya que ambas acciones se realizan en E3 (adelantamiento E3 $->$ E3).

- Dependencias de control:
  - En las instrucciones 5, 6 y 14 por salto condicional. El procesador debe introducir 2 ciclos de espera detrás de cada una de ellas, ya que no se sabe si el salto es efectivo o no hasta la etapa E3, en la que se evalúa la condición.
  - En las instrucciones 8 y 10 por salto incondicional. En este caso el procesador debe introducir 1 ciclo de espera detrás de cada una de ellas, ya que la dirección de salto se calcula en la etapa E2.

c) El CPI que se obtiene, utilizando adelantamientos, se calcula de la forma siguiente:

- Número de instrucciones ejecutadas (NI):

$$3 + N \times [2 + 0,35 \times 2 + 0,4 \times 2 + 0,25 \times 3 + 3] + 3 = 6 + N \times 7,25$$

- Número de ciclos empleados en la ejecución (NC):

$$3+N\times[(2+1dd+2dc)+0,35\times(2+1dc)+0,4\times(2+2dc)+0,25\times(3+3dc)+(3+2dc)]+3 = 6+N\times 14,15$$

donde dd y dc indican los parones debidos a dependencias de datos y de control respectivamente.

- Número de ciclos por instrucción:

$$CPI = \frac{NC}{NI} \approx \frac{14,15}{7,25} = 1,95$$

d) Se puede colocar la instrucción 13 entre la 4 y la 5. De este modo se elimina el único ciclo de parada debido a dependencia de datos existente, eliminándose en total N ciclos de parada, con lo que el nuevo CPI es $\approx 13,15/7,25 = 1,81$ y la ganancia obtenida $G = 1,95/1,81 = 1,077$

La ganancia no es significativa debido a que el mayor número de ciclos de espera se debe a las dependencias de control, y éstas no se solucionan con los adelantamientos.

e) Utilizando predicción dinámica solo se producen ciclos de espera cuando se falla en la predicción. Dado que esta comprobación se realizaría en la etapa E3, se introducirán en este caso 2 ciclos de espera. En la ejecución de la instrucción 14 se producen 2 fallos de predicción: la primera vez que se ejecuta (ya que se predice que no se salta y se salta), y la última (el bit de historia tiene el valor 1, se predice saltar, pero se sale del bucle), resultando una tasa de fallos de predicción de 2/10.000 (0,02 %, prácticamente nula).

Puesto que el número de ciclos de espera en la ejecución de la instrucción 14 sin predicción era Nx2=20.000 y con predicción es 2x2 = 4, se produce una reducción de 19.996 ciclos en la ejecución de esta instrucción. Se podría decir que prácticamente desaparecen.

1 *(1 punto) Enumere qué señales del bus de control y con qué finalidad debe gobernar un módulo de entrada/salida para realizar acceso directo a memoria (DMA).*

SOLUCIÓN

BR (Bus Request): se usa para solicitar los robos de ciclo y acceder directamente a memoria. La CPU le notifica que tiene los buses mediante BA (Bus Acknowledge).

MEMREQ (Memory Request): para marcar la temporización de los ciclos de acceso a memoria.

READ y WRITE: para indicar si el acceso a memoria es para una lectura o una escritura.

2 *(1 punto) Explique los criterios que se deben seguir para asignar las prioridades de interrupción en un computador. Indique también la utilidad de las interrupciones no enmascarables.*

SOLUCIÓN

El criterio principal consiste en asignar mayor prioridad al dispositivo que pide interrupciones con mayor frecuencia. Esta asignación de prioridades es óptima. Algunos periféricos excepcionalmente no siguen esta regla. Por ejemplo, algunos periféricos en los que se apoya el sistema operativo para realizar sus funciones: consola de operación, temporizadores programables, etc. Los periféricos que realizan sus operaciones mediante acceso directo a memoria no tienen urgencia puesto que solicitan interrupciones para notificar que el fin de las operaciones.

La línea de interrupciones no enmascarables se usa para notificar sucesos que no admiten demora en su tratamiento y suponen fallos de funcionamiento. Por ejemplo un fallo de energía.

3 *Sea un computador con un procesador de 64 bits con una capacidad de procesamiento de 2.000 MIPS en el que la secuencia de reconocimiento de interrupciones tiene una duración de 4 ns. A este computador están conectados los siguientes periféricos:*

- *Red Ethernet.*
  - *Velocidad de transmisión de 1 Gigabit por segundo (10^9 bits/s).*
  - *Bloques de longitud variable entre 64 y 1.536 bytes.*
  - *Registro de datos de 64 bits.*
  - *Las rutinas de inicio y fin de una operación de E/S tienen 100 y 200 instrucciones respectivamente.*
  - *La rutina de tratamiento de las interrupciones consta de 24 instrucciones.*
- *Disco duro.*
  - *Velocidad de transferencia: 10^8 bytes/s.*
  - *Tiempo medio de acceso: 4 ms.*
  - *Tamaño del sector: 512 bytes.*
  - *Buffer de 2 registros de datos de 64 bits.*
  - *Las rutinas de inicio y fin de una operación de E/S tienen 80 y 400 instrucciones respectivamente.*
  - *Operación por acceso directo a memoria (DMA).*

En este computador el protocolo de concesión y liberación de los buses dura un total de 2 ns y un ciclo de acceso a memoria dura 1 ns.

a) (1 punto) *Calcule cuántos módulos de red Ethernet podrían operar simultáneamente.*

b) (1 punto) *Calcule cuántos módulos de disco duro podrían operar simultáneamente.*

c) (2 puntos) *Calcule el tiempo total de una operación en la red Ethernet para transmitir un bloque de 1.5 kB así como el tiempo que se emplea de CPU.*

d) (2 puntos) *Calcule el tiempo total de una operación de lectura de un sector del disco duro así como el tiempo que se emplea de CPU.*

A partir del instante t=0, comienza la lectura de un fichero de 15 kB de la unidad de disco y, a medida que van estando disponibles sus datos, se transmiten simultáneamente por la red Ethernet en bloques de 1,5 kB.

e) (1 punto) *Calcule el instante t_{fin} en el que termina la transmisión del último bloque del fichero.*

f) (1 punto) *Calcule qué porcentaje de tiempo queda libre para la CPU durante el intervalo que comienza en t=0 y termina en t_{fin}.*

SOLUCIÓN

a) Se calcula la capacidad de procesamiento necesaria para atender las interrupciones de la red.

$$\begin{aligned} CP_{Ethernet} &= Freq_{int} \times I_{int} = \frac{10^9\ bits/s}{64\ bits} \times (4 \cdot 10^{-9}\ s \times 2 \cdot 10^9\ instr/s + 24\ instr) = \\ &= \frac{10^9\ bits/s}{64\ bits} \times 32\ instr = 500 \cdot 10^6\ instr/s = 500\ MIPS \end{aligned}$$

Se podrían atender a un máximo de 2.000 MIPS/500 MIPS = 4 unidades de red.

b) Se calcula la capacidad de procesamiento necesaria para atender los robos de ciclo del disco duro. Como tiene 2 registros de datos de 64 bits, por cada robo hará una ráfaga de 2 accesos a memoria.

$$t_{robo-ciclo} = 2\ ns + 2 \times 1\ ns = 4\ ns$$

$$\begin{aligned} CP_{HD} &= Freq_{robo-ciclo} \times I_{robo-ciclo} = \frac{10^8\ bits/s}{16\ bytes} \times (4 \cdot 10^{-9}\ s \times 2 \cdot 10^9\ instr/s) = \\ &= \frac{10^8\ bits/s}{16\ bytes} \times 8\ instr = 50 \cdot 10^6\ instr/s = 50\ MIPS \end{aligned}$$

Se podrían atender a un máximo de 2.000 MIPS/50 MIPS = 40 unidades de disco.

c)

$$\begin{aligned} t_{op-Ethernet} &= t_{ini} + t_{trans} + t_{última-int} = t_{ini} + t_{trans} + (t_{sri} + t_{rti} + t_{fin}) = \\ &= \frac{100\ instr}{2 \cdot 10^9\ instr/s} + \frac{1.536\ bytes \times 8\ bits/bytes}{10^9\ bits/s} + (4\ ns + \frac{24\ instr}{2 \cdot 10^9\ instr/s} + \frac{200 instr}{2 \cdot 10^9\ instr/s}) = \\ &= 50\ ns + 12.288\ ns + 4\ ns + 12\ ns + 100\ ns = 12.454\ ns \end{aligned}$$

$$\begin{aligned} t_{cpu-Ethernet} &= t_{ini} + (N_{int} - 1) \times t_{int} + t_{última-int} = t_{ini} + N_{int} \times (t_{sri} + t_{rti}) + t_{fin} = \\ &= 50\ ns + \frac{1.536\ bytes}{8\ bytes} \times (4\ ns + 12\ ns) + 100\ ns = 50\ ns + 3.072\ ns + 100\ ns = 3.222\ ns \end{aligned}$$

d)

$$\begin{aligned} t_{op-HD} &= t_{ini} + t_{acc} + t_{trans} + t_{último-robo-ciclo} + t_{int} = \\ &= t_{ini} + t_{acc} + t_{trans} + t_{último-robo-ciclo} + (t_{sri} + t_{fin}) = \\ &= \frac{80\ instr}{2 \cdot 10^9\ instr/s} + 4\ ms + \frac{512\ bytes}{10^8\ bytes/s} + 4\ ns + (4\ ns + \frac{400 instr}{2 \cdot 10^9\ instr/s}) = \\ &= 40\ ns + 4 \cdot 10^6\ ns + 5.120\ ns + 4\ ns + (4\ ns + 200\ ns) = 4.005.368\ ns \end{aligned}$$

$$\begin{aligned} t_{cpu-HD} &= t_{ini} + N_{robo-ciclo} \times t_{robo-ciclo} + t_{int} = \\ &= 40\ ns + \frac{512\ bytes}{2 \cdot 8\ bytes} \times 4\ ns + (4\ ns + 200\ ns) = 40\ ns + 128\ ns + 204\ ns = 372\ ns \end{aligned}$$

e) Para leer del disco y transmitir por la Ethernet en bloques de 1,5 kB un fichero de 15 kB, se leen sus 30 sectores del disco y se transmiten 10 bloques por la red Ethernet. Sin embargo, hay que tener en cuenta que cada vez que se leen 3 sectores del disco se procede a transmitir el correspondiente bloque de 1,5 kB por la red Ethernet.

A partir de los resultados de apartados anteriores se comprueba que una operación en la red Ethernet es más corta que 3 operaciones en el disco. Por lo tanto, la red está siempre disponible para transmitir el siguiente bloque y así las operaciones en el disco duro se realizan consecutivamente.

De modo que la transmisión del último bloque del fichero comienza cuando finaliza la trigésima operación del disco duro y el instante t_{fin} en el que termina será:

$$t_{fin} = 30 \times t_{op-HD} + t_{op-Ethernet} = 30 \times 4.005.368\ ns + 12.454\ ns = 120.173.494\ ns = 120.173,494\ \mu s$$

f) El tiempo total empleado por la CPU es el empleado en 30 operaciones del disco y 10 operaciones de la red Ethernet.

Cada 3 disco, 1 red.

$$t_{total-cpu} = 30 \times t_{cpu-HD} + t_{cpu-Ethernet} = 30 \times 372\ ns + 10 \times 3.222\ ns = 43.380\ ns = 43,380\ \mu s$$

Así el porcentaje de tiempo libre de la CPU es:

$$\%_{cpu-libre} = \frac{t_{total} - t_{ocupada}}{t_{total}} \times 100 = \frac{120.173,494\ \mu s - 43,380\ \mu s}{120.173,494\ \mu s} \times 100 = 99,96\ \%$$

1 *(1 punto) Explique los tipos de proximidad de referencias (locality) que existen en la ejecución de los programas.*

SOLUCIÓN

Existen dos tipos de proximidad que coexisten normalmente, en mayor o menor medida, en la ejecución de todo programa:

Temporal. Cuando se tiende a hacer referencia a direcciones a las que se ha accedido en un pasado reciente. En el caso de un bucle, se accede nuevamente a la dirección de comienzo del bucle tras la instrucción de salto.

Espacial. Cuando se tiende a hacer referencia a direcciones cercanas a las que se ha accedido en un pasado reciente. Por ejemplo, cuando se accede a los elementos de un vector o matriz. Un caso particular es la proximidad **Secuencial**, que se da cuando se accede a direcciones consecutivas. Este último es el caso del acceso secuencial a las instrucciones.

2 *(1 punto) Se está ejecutando en un computador con una memoria caché de datos de ubicación directa el siguiente fragmento de un programa:*

```
for (i=0; i<2048; i++)  /* el índice i se almacena en un registro */
   b[i] = a[i];
```

La memoria caché tiene 512 bloques de 16 bytes cada uno y los vectores `a` *y* `b` *tienen 2048 elementos de 32 bits cada uno. Enumere los tipos de fallos de caché y explique cuáles se pueden producir y bajo qué circunstancias suponiendo que la memoria caché está inicialmente invalidada.*

SOLUCIÓN

Existirán siempre fallos de primera referencias al acceder a los bloques que ocupan los vectores.

No se producen fallos de capacidad ya que únicamente se necesitan 2 bloques de memoria caché para contener el bloque de `a` y el bloque de `b` que se está accediendo.

Únicamente existirán fallos por conflicto si a los vectores `a` y `b` les corresponden los mismos bloques de cache.

3 *Se tiene un computador de 32 bits con memoria virtual paginada y cachés separadas para instrucciones y datos. Las memorias cachés tienen las siguientes características:*

- *Capacidad de cada memoria caché: 32 KB.*
- *Tamaño de los bloques de caché: 32 bytes.*
- *Organización asociativa por conjuntos de 4 bloques.*
- *Política de reemplazo LRU* (Least Recently Used).
- *Política de lectura: OOF* (Out of Order Fetch).
- *Política de escritura de la caché de datos: diferida con actualización* (CBWA: Copy Back With Allocation). *En caso de fallo se escribe primero en memoria principal y posteriormente se lleva el bloque a la memoria caché.*
- *Tiempo de acceso 2 ns.*

Las características de la memoria virtual son las siguientes:

- *Tamaño de las páginas de 8 KB.*

- *TLBs separadas, asociativas con 16 entradas y tiempo de acceso de 1 ns.*
- *Tres niveles de tablas de páginas.*
- *Cada tabla de páginas ocupa 1 página.*
- *Cada entrada de las tablas de páginas tiene 1 palabra.*

a) (1 punto) Indique el formato de las direcciones virtuales y el espacio virtual direccionable.

b) (1 punto) Indique si es posible acceder simultáneamente a las TLBs y a las cachés.

c) (1 punto) Calcule el tiempo máximo de acceso a este sistema de memoria teniendo en cuenta que el tiempo de acceso a memoria principal es 40 ns y el tiempo necesario para leer o escribir un bloque es 60 ns. (Suponga que no hay fallos de página.)

En este computador se ha ejecutado el siguiente fragmento de un programa:

```
for (i=0; i<50; i++)
  for (j=0; j<50; j++)
    for (k=0; k<50; k++)
      c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

d) (1 punto) Calcule el número de páginas que ocupan las matrices así como el número de páginas que se necesitan para su traducción si cada elemento ocupa una palabra y están ubicadas a partir de las direcciones H'1000000, H'2000000 y H'3000000.

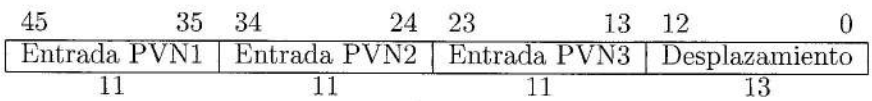
e) (1 punto) Calcule el número de bloques de caché que ocupan las matrices y el número de fallos en la caché de datos que produce el fragmento de código si la caché está inicialmente invalidada.

f) (1 punto) Calcule las tasas de aciertos en la caché y en la TLB de datos que produce el fragmento de código suponiendo que la TLB está también inicialmente invalidada.

g) (2 puntos) Calcule el tiempo total de acceso y ocupación del sistema de memoria debido a los accesos a datos que produce el fragmento de código.

SOLUCIÓN

a) Si las páginas son de 8 KB y las entradas de las tablas de páginas son de una palabra (4 B), las tablas de página tendrán 2 Kentradas (2.048). De este modo las direcciones virtuales se interpretan así:

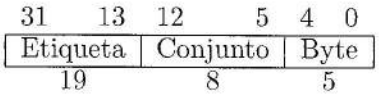


El espacio virtual direccionable es de 2^{46} bytes, es decir 64 TB.

b) Como es asociativa por conjuntos habrá que calcular el número de conjuntos.

$$Núm\_conjuntos = \frac{32\ KB}{4\ Bq/Conjunto \cdot 32\ bytes/Bq} = 256\ Conjuntos$$

Por lo tanto las direcciones físicas se interpretan así por la unidad de control de la memoria caché:



Se pueden solapar los accesos a las TLBs y memorias caché porque el campo desplazamiento, que no se traduce, es igual a los campos conjunto y byte.

c) Se calculará el tiempo máximo de traducción y el tiempo máximo de acceso a la información.

El tiempo máximo de traducción se produce cuando hay un fallo en la TLB y hay que traducir en los 3 niveles de tablas de páginas de memoria principal.

$$t_{máximo\_traducción} = 1\ ns + 3 \times 40\ ns = 121\ ns$$

El tiempo máximo de acceso a la información se produce en la lectura de un dato, cuando hay fallo en la memoria caché y el bloque a desalojar está modificado:

$$t_{máximo\_información} = 2\ ns + 60\ ns + 40\ ns = 102\ ns$$

Así se obtiene:

$$t_{máximo\_acceso} = 121\ ns + 102\ ns = 223\ ns$$

d) Se calcula el número de páginas que ocupa cada matriz:

$$Núm\_págs/matriz = \frac{50 \times 50 \times 4\ bytes}{8\ KB/págs} = 1,22$$

Lo que supone que como cada matriz está alineada a página, se necesitan 2 páginas para los elementos de cada matriz, es decir, 6 páginas. Por otra parte, las direcciones virtuales se interpretan así:

45 35	34 24	23 13	12 0
Entrada PVN1	Entrada PVN2	Entrada PVN3	Desplazamiento
00000000000	00000000001	00000000000	0000000000000
00000000000	00000000010	00000000000	0000000000000
00000000000	00000000011	00000000000	0000000000000
11	11	11	13

Es decir, se necesita además de la tabla de páginas de nivel 1, la primera tabla de páginas de nivel 2 (0) y 3 tablas de páginas de nivel 3 (1, 2 y 3). Es decir, 5 tablas de páginas que ocupan otras 5 páginas.

e) Se calcula el número de bloques que ocupa cada matriz:

$$Núm\_bloques/matriz = \frac{50 \times 50 \times 4\ bytes}{32\ bytes/bloque} = 312,5\ bloques$$

Como cada matriz está alineada a bloque, se necesitan 313 bloques para cada matriz. Es decir, 939 bloques para las 3 matrices. Como hay 1.024 bloques no habrá fallos por capacidad y como hay 4 bloques por conjunto no hay fallos por conflicto. Así que sólo habrá 939 fallos forzosos.

f) A partir de los resultados anteriores se calculan las tasas de acierto (Hr). El número total de accesos a memoria es $50 \times 50 \times 50 \times 4 = 500.000$.

$$Hr_{caché} = \frac{500.000 - 939}{500.000} = 0,9981$$

$$Hr_{TLB} = \frac{500.000 - 6}{500.000} = 0,9999$$

g) Como se solapan los accesos a las TLBs y memorias caché, no se puede calcular separadamente el tiempo de traducción y el tiempo de acceso la información. Habrá que considerar los casos posibles, aunque hay que tener

en cuenta que los fallos de memoria caché son todos forzosos y que, como el contenido original está invalidado, no se reemplazan bloques modificados.

De los 500.000 accesos, 6 accesos producen fallo en TLB y en memoria caché. Para estos casos, los tiempos de acceso y ocupación se calculan:

$$t_{acc} = T_{TLB} + 3 \times T_{MP} + T_{caché} + T_{MP} = 1\ ns + 3 \times 40\ ns + 2\ ns + 40\ ns = 163\ ns$$

$$t_{ocup} = T_{TLB} + 3 \times T_{MP} + T_{caché} + T_{bl-MP} = 1\ ns + 3 \times 40\ ns + 2\ ns + 60\ ns = 183\ ns$$

De los 499.994 accesos que aciertan en TLB, 933 accesos (939 - 6) producen fallo en memoria caché. Para estos casos, los tiempos de acceso y ocupación se calculan:

$$t_{acc} = T_{caché} + T_{MP} = 2\ ns + 40\ ns = 42\ ns$$

$$t_{ocup} = T_{caché} + T_{bl-MP} = 2\ ns + 60\ ns = 62\ ns$$

Por último, hay 499.061 accesos (499.994 - 933) que producen acierto en TLB y en memoria caché. Para estos casos, los tiempos de acceso y ocupación se calculan:

$$t_{acc} = T_{caché} = 2\ ns$$

$$t_{ocup} = T_{caché} = 2\ ns$$

Así que los tiempos totales de de acceso y ocupación resultantes son:

$$t_{acc} = 6 \times 163\ ns + 933 \times 42\ ns + 499.061 \times 2\ ns \approx 1,038\ ms$$

$$t_{ocup} = 6 \times 183\ ns + 933 \times 62\ ns + 499.061 \times 2\ ns \approx 1,0578\ ms$$

1 *Sea un procesador con tamaño de palabra de 32 bits y cuyo sistema de memoria tiene las siguientes características:*

- *Memoria principal con entrelazado simple de orden inferior con 8 módulos y tiempo de acceso de 40 ns.*
- *Memorias caché de nivel 1 (L1) separadas para instrucciones y datos. Las características de la caché de datos (McaD) son las siguientes:*
  - *Capacidad 16KB, bloques de 32B y tiempo de acceso 2ns.*
  - *Ubicación asociativa por conjuntos de 4 bloques, y política de reemplazo LRU dentro del conjunto.*
  - *Política de lectura out of order fetch.*
  - *Política de escritura aplazada con actualización (CBWA). En los fallos de escritura, se escribe primero en Mp y posteriormente se lleva el bloque actualizado a la McaD.*

a) Indique razonadamente cómo interpreta la McaD las direcciones físicas.

b) En este procesador se va a ejecutar un programa, del que se ha extraido un fragmento de código, para el que se plantean 2 versiones:

- Versión 1 -

```
for (i=0; i<1024; i++){
   C[i] = B[i] + A[i];
   G[i] = r + x*F[i];
}
```

- Versión 2 -

```
for (i=0; i<1024; i++)
  C[i] = B[i] + A[i];
for (i=0; i<1024; i++)
  G[i] = r + x*F[i];
```

Sabiendo que:

- *Las variables i, r y x están asignadas a registros del procesador.*
- *Los vectores utilizados son de 1.024 elementos de 1 palabra.*
- *La dirección de Mp a partir de la que está almacenado el vector A es H´01000, y los vectores B, C, F y G están almacenados, en este orden, a continuación de A.*
- *La McaD está inicialmente invalidada.*

b.1) Calcule el número de bloques que ocupan los vectores.

b.2) Especifique, de forma razonada, en qué conjuntos de la McaD se alojarán los diferentes vectores.

b.3) Para la versión 1 del código, rellene la siguiente tabla, para las dos primeras iteraciones del bucle.

Acceso a	*A[0]*	*B[0]*	*C[0]*	*F[0]*	*G[0]*	*A[1]*	*B[1]*	*C[1]*	*F[1]*	*G[1]*
Acierto/Fallo										
Lectura/Escritura										
Reemplazo de:										

b.3.1) *Calcule la tasa de aciertos de la McaD obtenida en la ejecución completa de este bucle.*

b.3.2) *Calcule el tiempo medio de acceso empleado en los accesos a los vectores.*

b.4) Calcule la tasa de aciertos de la McaD, así como el tiempo medio de acceso empleado en los accesos a los vectores en la versión 2.

b.5) Calcule con cuál de las versiones se obtendría un menor tiempo de ejecución, sabiendo que en la versión 2 se ejecutan 3 instrucciones más por iteración (debidas a la consulta y actualización de la variable de control del bucle), y cada una tarda 8 ns en ejecutarse.

c) Suponga ahora que el procesador tiene memoria virtual paginada, con las siguientes características:

- *Espacio de direcciones lógico de 4TB y páginas de 4KB*
- *3 niveles de tablas de páginas, cuyas entradas ocupan 1 palabra, y una TLB con tiempo de acceso de 1ns*

c.1) Describa el formato de las direcciones virtuales, así como el tamaño de cada uno de sus campos.

c.2) Calcule el número de páginas que ocupan los vectores, así como las tablas de páginas necesarias para su traducción y las entradas de cada tabla utilizadas, suponiendo que el vector A comienza en la dirección virtual H'0100000000, y el resto de los vectores están almacenados a continuación.

c.3) Determine si es posible acceder a la vez a la TLB y la McaD. Considerando su respuesta y las características de la McaD descritas en los apartados anteriores, calcule el tiempo mínimo y máximo de acceso a la información para la versión 2 del código suponiendo que no se produce fallo de página. Indique en qué accesos se consigue el mínimo y en cuáles el máximo tiempo de acceso.

SOLUCIÓN

a) Como la McaD tiene una capacidad de 16KB y los bloques son de 32B (8 palabras), la McaD está compuesta por: $16KB \ / \ 32B/bloque = 2^9 = 512 \ bloques$. Al estar organizada en conjuntos de 4 bloques, tiene un total de $512/4 = 128$ conjuntos. Por lo tanto, la McaD interpreta las direcciones físicas, suponiendo que estas son de 32 bits, de la siguiente forma:

etiqueta	conjunto	byte en bloque
20 bits	7 bits	5 bits

b) Resultados para las dos versiones del código planteadas

b.1) El vector A comienza en una dirección alineada a bloque (los 5 bits menos significativos son cero) y consta de 1.024 elementos de 4B cada uno, por lo que ocupa:

$(1.024 \ elementos \times 4B/elemento) \ / \ 32B/bloque = 128 \ bloques.$

Como el resto de los vectores están almacenados a continuación de A y tienen la misma dimensión, en total los 5 vectores ocupan $128 \times 5 = 640$ bloques de Mp consecutivos.

b.2) La dirección de Mp donde comienza de A (H´01000), es interpretada por la McaD de la siguiente forma:

000 ... 001	0000000	00000
etiqueta	conjunto	byte

El primer bloque de A se alojará por lo tanto en el conjunto 0, el segundo en el conjunto 1, y así sucesivamente hasta el último bloque, que se alojará en el último conjunto de la McaD (conjunto 127), ocupando uno de los 4 bloques de cada conjunto. Puesto que el resto de los vectores ocupan el mismo número de bloques, y están almacenados en Mp justo a continuación de A, a todos los vectores les corresponden los mismos conjuntos de la McaD, de nuevo del 0 al 127.

b.3) El comportamiento en las dos primeras iteraciones del bucle, teniendo en cuenta que los 5 bloques referenciados corresponden al mismo conjunto, y que la política de reemplazo dentro del conjunto es LRU, es el siguiente:

Acceso a	A[0]	B[0]	C[0]	F[0]	G[0]	A[1]	B[1]	C[1]	F[1]	G[1]
Acierto/Fallo	F	F	F	F	F	F	F	F	F	F
Lectura/Escritura	L	L	E	L	E	L	L	E	L	E
Reemplazo de:	-	-	-	-	A[0]	B[0]	C[0]	F[0]	G[0]	A[1]

Los cinco primeros accesos son fallos de primera referencia, ya que la McaD estaba inicialmente invalidada. Además, el fallo debido a la referencia a G[0] provoca el desalojo de uno de los 4 bloques del conjunto en el que le corresponde ubicarse (conjunto 0), bloques ocupados por el primer bloque de los vectores A, B, C y F respectívamente. El bloque eligido, el menos recientemente utilizado, es el ocupado por A[0] .. A[7].

Los accesos correspondientes a la segunda iteración producen fallo, dado que los bloques correspondientes ya se llevaron a la McaD y fueron reemplazados, en el orden indicado en la tabla, por coincidir en el mismo conjunto y haber sólo 4 bloques por conjunto.

b.3.1) El comportamiento analizado en el apartado anterior para las dos primeras iteraciones se repite para las restantes, por lo que la tasa de aciertos de la McaD es del 0 %

b.3.2) Puesto que todos los accesos fallan en la McaD, todos tardan el mismo tiempo:

$$T_{McaD} + T_{Mp} = 2 + 40 = 42 \ ns$$

Esto es así ya que la política de lectura es *out of order fetch* y además no se producen reemplazos de bloques modificados (cuando se falla en escritura primero se escribe en Mp y después se lleva el bloque ya modificado a caché).

b.4) En el primer bucle se referencian únicamente 3 vectores por lo que, aunque coinciden en el mismo conjunto, al haber 4 bloques por conjunto no se desalojan uno a otro, produciéndose únicamente fallos de primera referencia, en total 128 x 3 = 384 fallos.

En el segundo bucle se referencian únicamente los otros 2 vectores, por lo que siguiendo el mismo razonamiento anterior se producirán únicamente fallos de primera referencia, en total 128 x 2 = 256 fallos, que sumados a los del primer bucle dan un total de 640 fallos.

La tasa de aciertos, puesto que solo se producen fallos de primera referencia y los bloques tienen capacidad para 32B (8 palabras) es $7/8 = 0{,}875 = 87{,}5\,\%$.

Para calcular el tiempo medio de acceso hay que tener en cuenta que, puesto que los 5 vectores no caben en la McaD, cabría la posibilidad de que en el segundo bucle se tuvieran que reemplazar bloques modificados, correspondientes a C, para alojar algunos de los bloques de F y G. No obstante, dado que la política de reemplazo utilizada es LRU, los bloques que se reemplazarán serán los de A y B antes que los de C, que son los últimos que se referencian en el primer bucle. En resumen, la probabilidad de tener que reemplazar un bloque modificado es cero.

Teniendo en cuenta lo anterior y que el número total de accesos a la McaD es $1.024 \times 5 = 5.120$ *accesos*, los tiempos total y medio (o efectivo) empleados en los accesos a los vectores son los siguientes:

$$T_{total} = (5.120 - 640) \times 2\ ns + 640 \times (2 + 40)\ ns = 8.960 + 26.880 = 35.840\ ns$$

$$T_{ef} = 35.840/5.120 = 7\ ns$$

b.5) En la versión 1, el tiempo total empleado en los accesos a datos es: $5.120\ accesos \times 42\ ns = 215.040\ ns$

En la versión 2, el tiempo adicional debido a la ejecución de 3 instrucciones más por iteración es:

$$3\ instrucciones/iteracion \times 8\ ns/instruccion \times 1.024\ iteraciones = 24.576\ ns,$$

que sumado al tiempo calculado en el apartado anterior da un total de: $35.840 + 24.576 = 60.416\ ns$

La versión con la que se conseguiría un menor tiempo de ejecución sería por lo tanto la versión 2.

c) El espacio de direcciones lógico es de 4 TB (2^{42} bytes), por lo que las direcciones virtuales tienen una longitud de 42 bits.

c.1) Al ser las páginas de 4 KB (2^{12} bytes), se necesitan 12 bits para identificar cualquier byte dentro de una página. Los 30 bits restantes de la dirección se utilizarán para indexar los tres niveles de tablas de páginas.

En este caso, la división razonable es utilizar 10 bits para cada nivel, ya que de esta forma todas las tablas de páginas ocupan 1 página ($2^2\ bytes\ por\ entrada \times 2^{10}\ entradas\ por\ tabla = 2^{12}\ bytes$), con lo que el formato de las direcciones virtuales sería el siguiente:

n1	n2	n3	byte
10 bits	10 bits	10 bits	12 bits

c.2) El vector A comienza en una dirección virtual (`H'0100000000`) alineada a página, ya que los 12 bits menos significativos son cero, por lo que ocupa:

$$(1.024\ elementos \times 4B/elemento)\ /\ 2^{12}B/pagina = 1\ \text{página}$$

Como el resto de los vectores están almacenados a continuación y tienen el mismo número de elementos, los 5 vectores ocupan un total de 5 páginas. Para saber las tablas y entradas utilizadas para su traducción hay que tener en cuenta la dirección virtual correspondiente al vector A.

La dirección `H'0100000000`, corresponde al primer byte de la página (últimos 12 bits a cero), y los índices a los tres niveles de tablas de páginas a los valores: n1 = 1, n2 = 0 y n3 = 0. Se necesitarán por lo tanto una tabla de cada nivel para realizar la traducción, de las que se utilizarán la entrada 1 de la tabla de primer nivel, la entrada 0 de la de segundo nivel y las entradas 0 a 4 de la de tercer nivel (para traducir las 5 páginas consecutivas ocupadas por los vectores).

c.3) Para determinar si se puede solapar el acceso a la TLB y a la McaD hay que comprobar si con la parte de la Dv que no se traduce (los 12 bits menos significativos) se pueden direccionar todos los conjuntos y los bytes dentro de un bloque de la McaD. En este caso sí sería posible ya que se necesitan 7 bits para seleccionar el conjunto más 5 para el byte dentro del bloque.

El tiempo mínimo de acceso a la información se da cuando hay acierto en la TLB y en la McaD:

$$T_{min} = max(T_{TLB}, T_{Mca}) = T_{Mca} = 2\ ns$$

El tiempo máximo de acceso se da cuando hay fallo en la TLB, hay que consultar los 3 niveles de tablas de páginas residentes en Mp para realizar la traducción, hay fallo en la McaD y hay que acceder a Mp. Según se vió en el apartado b.3.2, no hay que considerar el tiempo empleado en reemplazar un bloque modificado ya que no se puede dar esta circunstancia.

$$T_{max} = T_{TLB} + T_{TP's} + T_{Mca} + T_{Mp} = 1 + 3 \times 40 + 2 + 40 = 163\ ns$$

Obsérvese que el tiempo máximo es el mismo para los accesos de lectura y de escritura ya que, según el enunciado, cuando se produce un fallo de escritura en la McaD primero se escribe en Mp y a continuación se envía el bloque, ya actualizado, a la McaD.

1 *Sea un computador cuyo procesador dispone de un pipeline de 5 etapas en el que se ejecutan independientemente dos fragmentos de programa que realizan la misma función (se trata en realidad de dos implementaciones diferentes de una misma función). La operación realizada en ambos casos consiste en separar los elementos que ocupan las posiciones pares de un vector de entrada* `vect`*, de los elementos que ocupan las posiciones impares de ese mismo vector, dejando el resultado en un par de vectores de salida:* `vpar` *y* `vimpar`*.*

El pipeline del procesador introduce ciclos de parada para resolver las dependencias de control, dispone de todo tipo de mecanismos de adelantamiento y emplea las siguientes etapas:

- *E1: Fetch e incremento del PC.*
- *E2: Decodificación, lectura de registros, evaluación de la condición y cálculo de la dirección destino en las instrucciones de salto.*
- *E3: Ejecución y cálculo de la dirección para las instrucciones load y store.*
- *E4: Acceso a memoria para lectura o escritura de datos.*
- *E5: Escritura en registros.*

A continuación se muestra el código de los fragmentos mencionados:

Implementación 1 (1.000 iteraciones).

```
(1) PROG1: add r4, r0, r0
(2)        add r10, r0, r0
(3)        add r5, r0, r0
(4)        add r11, r0, r0
(5) iter:  mask r6, r5, 1
(6)        cmp r7, r6, 1
(7)        beq r7, $impar
(8)        ld r8, vect(r11)
(9)        st r8, vpar(r10)
(10)       br $comun
(11) impar: ld r8, vect(r11)
(12)       st r8, vimpar(r10)
(13)       add r10, r10, 4
(14) comun: add r11, r11, 4
(15)       add r5, r5, 1
(16)       cmp r7, r5, r9
(17)       blt r7, iter
```

Implementación 2 (500 iteraciones).

```
(1) PROG2: add r4, r0, r0
(2)        add r10, r0, r0
(3)        add r5, r0, r0
(4)        add r11, r0, r0
(5) iter2: ld r8, vect(r11)
(6)        st r8, vpar(r10)
(7)        add r11, r11, 4
(8)        ld r8, vect(r11)
(9)        st r8, vimpar(r10)
(10)       add r10, r10, 4
(11)       add r11, r11, 4
(12)       add r5, r5, 2
(13)       cmp r7, r5, r9
(14)       blt r7, iter2
(15)       add r5, r0, r0
```

Se pide lo siguiente:

a) *Determine para ambos programas en qué instrucciones se producen ciclos de parada por dependencias de datos o por dependencias de control, especificando en cada caso cuántos ciclos de parada se consumen.*

a.1) *Calcule el nº de ciclos invertido en la ejecución de cada uno de los dos programas, así como su respectivo CPI. Considere para ello que el vector* `vect` *consta de 1.000 elementos, es decir, que el valor inicial de* `r9` *es 1.000.*

a.2) *Sabiendo que ambos programas realizan la misma función, calcule la ganancia que se obtiene al ejecutar la segunda implementación del programa,* `PROG2`*, en lugar de la primera (*`PROG1`*).*

b) *Suponga ahora que el procesador dispone de un predictor de salto de 1 bit que actúa en la etapa "E1:Fetch" e inicialmente predice que no se salta.*

b.1) *Determine para este procesador qué saltos se predicen con acierto y cuáles con fallo en ambos programas.*

b.2) *Calcule el nº de ciclos invertido por cada programa y su correspondiente CPI para este nuevo procesador.*

c) *Justifique si el programa* `PROG2` *funcionaría correctamente en el caso de que el procesador utilizase salto retardado con* delay-slot-1*. Si no fuese así, describa qué modificaciones debería realizar para lograr el funcionamiento previsto sin aumentar innecesariamente el número de ciclos empleados en la ejecución.*

SOLUCIÓN

a) Puesto que el procesador utilizado dispone de todo tipo de mecanismos de adelantamiento, muchas de las dependencias de datos presentes en ambos programas se resuelven sin que se añada ningún ciclo de parada. Las dependencias de control se producen lógicamente en las instrucciones de salto, obligando a consumir un ciclo de parada en cada uno de ellos ya que todas las operaciones relativas a saltos se realizan en la etapa 2 del pipeline. La siguiente tabla resume todas las dependencias que producen ciclos de parada en el procesador considerado:

Programa	Instrucciones	Registro	Tipo	Ciclos de parada
PROG1	(7)	–	control	1 c.
PROG1	(10)	–	control	1 c.
PROG1	(17)	–	control	1 c.
PROG2	(14)	–	control	1 c.
PROG1	(6) → (7)	r7	datos: E3 → E2	1 c.
PROG1	(16) → (17)	r7	datos: E3 → E2	1 c.
PROG2	(13) → (14)	r7	datos: E3 → E2	1 c.

a.1) En cualquiera de los dos programas se ejecutan cuatro instrucciones antes de entrar en el bucle que trata los distintos elementos del vector `vect`. Además, el número de ciclos consumido en cada iteración (es decir, para cada elemento del vector de entrada), debido únicamente a la ejecución de las instrucciones de cada programa será:

PROG1:

Instrucciones ejecutadas al tratar con un elemento que ocupa posición impar:

`i5,i6,i7,i11,i12,i13,i14,i15,i16,i17`. *Total* : 10 *instrucciones*

Instrucciones ejecutadas al tratar con un elemento que ocupa posición par:

`i5,i6,i7,i8,i9,i10,i14,i15,i16,i17`. *Total* : 10 *instrucciones*

PROG2:

Instrucciones ejecutadas al tratar con cualquier elemento (ocupe este posición par o impar):

`i5,i6,i7,i8,i9,i10,i11,i12,i13,i14`. *Total* : 10 *instrucciones*

En total, se ejecutan $4 + 10 \times 1.000 = 10.004$ instrucciones en el programa PROG1 y $4 + 10 \times 500 + 1 = 5.005$ instrucciones en PROG2 (en este último se ejecuta la mitad de iteraciones, ya que en cada una de ellas se tratan dos elementos del vector).

Además, en el programa PROG1 se producen dos ciclos de parada por cada dependencia de datos en cada iteración, otros dos ciclos de parada por dependencias de control en las iteraciones impares y tres ciclos de parada por dependencias de control en las iteraciones pares. Todo ello lleva a consumir un total de:

PROG1: $4(i) + 1.000\ \textit{iteraciones} \times (0,5 \times (10(i) + 2(d) + 2(c)) + 0,5 \times (10(i) + 2(d) + 3(c))) = \ 14.504\ \textit{ciclos}$

donde (i) representa "ciclos debidos a ejecución de instrucciones", (d) son los ciclos debidos a las dependencias de datos y (c) los debidos a las dependencias de control.

En el caso del programa PROG2, los resultados obtenidos son los siguientes:

PROG2: $4(i) + 500\ \textit{iteraciones} \times (10(i) + 1(d) + 1(c)) + 1(i) = \ 6.005\ \textit{ciclos}$

El CPI obtenido en cada caso es:

PROG1: $14.504\ \textit{ciclos}/10.004\ \textit{instrucciones} \ = \ 1,45$

PROG2: $6.005\ \textit{ciclos}/5.005\ \textit{instrucciones} \ = \ 1,2$

a.2) Al ejecutar la segunda versión, PROG2, se logra una ganancia en CPI de 1,45/1,2 = 1,21. Sin embargo, la verdadera ganancia se obtiene al considerar los ciclos empleados, ya que el número de instrucciones ejecutadas en ambos programas es muy diferente:

$\textit{Ganancia} \ = \ 14.504 \ / \ 6.005 \ = \ 2,42$

b) El uso de un predictor de saltos no altera el número de instrucciones que se ejecutan ni afecta a las dependencias de datos, por lo que estas serán idénticas a las del apartado anterior. Sin embargo, en las dependencias de control el comportamiento será distinto. Si se acierta en la predicción no se consume ningún ciclo de espera.

En los fallos de predicción se consume un ciclo, igual que en el procesador que no dispone de predictor de saltos.

Concretando en las instrucciones de salto de cada programa:

- En el programa `PROG1` hay tres instrucciones de salto: la 10, correspondiente a un salto incondicional, y la 7 y la 17, que son saltos condicionales:
  - En la instrucción 10 siempre se consume un ciclo de espera, ya que no ha cambiado nada frente al procesador que no utiliza predictor de saltos. Puesto que se pasa 500 veces por esta instrucción, se consume un total de 500 ciclos de espera.
  - En la instrucción 7 se produce inicialmente un acierto, ya que la primera predicción corresponde salto no efectivo y primer elemento del vector que se analiza ocupa una posición par (posición 0). Todas las demás predicciones serán fallos, ya que se en cada nuevo paso por esa instrucción cambia la condición del salto entre "se toma" y "no se toma", por lo que las predicciones, que siempre se refieren a la situación observada en la iteración anterior, son erróneas. En total, se produce un acierto y 999 fallos, totalizando 999 ciclos de espera.
  - En la instrucción 17 se realiza el salto en todas las iteraciones menos en la última, por lo que se produce un fallo de predicción en la primera iteración (está inicializado a "no salto"), se acierta en las 998 siguientes y se falla en la última iteración, consumiéndose un total de 2 ciclos de espera.
- En el programa `PROG2` hay una única instrucción de salto, la 14, correspondiente a un salto condicional. Esta instrucción se ejecuta 500 veces y, por las mismas razones que la instrucción 17 en el `PROG1`, se producen 2 fallos y 498 aciertos, consumiéndose en total 2 ciclos.

b.1) A partir de lo expresado en el apartado anterior, se obtienen los siguientes valores para el número de ciclos empleados en la ejecución de cada programa y su correspondiente CPI:

`PROG1`: $4(i) + 1.000\ \textit{iteraciones} \times (0,5 \times (10(i) + 2(d)) + 0,5 \times (10(i) + 2(d))) + 500 + 999 + 2 = \ 13.505\ \textit{ciclos}$

`PROG2`: $4(i) + 500\ \textit{iteraciones} \times (10(i) + 1(d)) + 2 + 1 = \ 5.507\ \textit{ciclos}$

El CPI obtenido en cada caso es:

`PROG1`: $13.505\ \textit{ciclos}/10.004\ \textit{instrucciones} \ = \ 1,35$

`PROG2`: $5.507\ \textit{ciclos}/5.005\ \textit{instrucciones} \ = \ 1,1$

De nuevo se aprecia que la reducción en el número de ciclos, que es la verdaderamente importante, es mucho más significativa que la reducción en el valor del CPI.

c) En caso de utilizar *delay-slot-1* el programa `PROG2` no funcionaría correctamente, ya que se ejecutaría la instrucción (15) siempre que se ejecuta la (14), con lo que se pondría a cero el contador de iteraciones y el programa quedaría en un bucle infinito. Para solucionarlo podría ponerse una instrucción "NOP" tras la instrucción (14), pero ello aumentaría en 500 el número de ciclos de ejecución, ya que se ejecutaría una instrucción adicional (e inútil) en cada iteración. La solución adecuada consistiría en trasladar a la posición posterior al salto una de las instrucciones útiles que se ejecuta en cada iteración, de modo que no se afecte al resultado, por ejemplo la instrucción (10), con lo que la ejecución de una iteración quedaría como sigue:

`i5,i6,i7,i8,i9,i11,i12,i13,i14,i10.`

2 *Sea un multiprocesador de memoria compartida con 16 procesadores, conectados a través de un bus a la memoria principal. Cada procesador dispone de una memoria cache privada de datos, que por simplificar se va a considerar que tiene sólo cuatro bloques, es de ubicación directa y que cada bloque tiene sólo dos palabras.*

Para mantener la coherencia usa la política de invalidación MESI, con cuatro estados, sobre una implementación snoopy

Indique las acciones que desencadenan las siguientes operaciones y qué cambios se producen tanto en las cachés como en la memoria principal. Suponga que cada operación es independiente de las demás, por lo que se parte del estado inicial mostrado en la siguiente figura. Para simplificar la figura, el tag contiene la dirección completa.

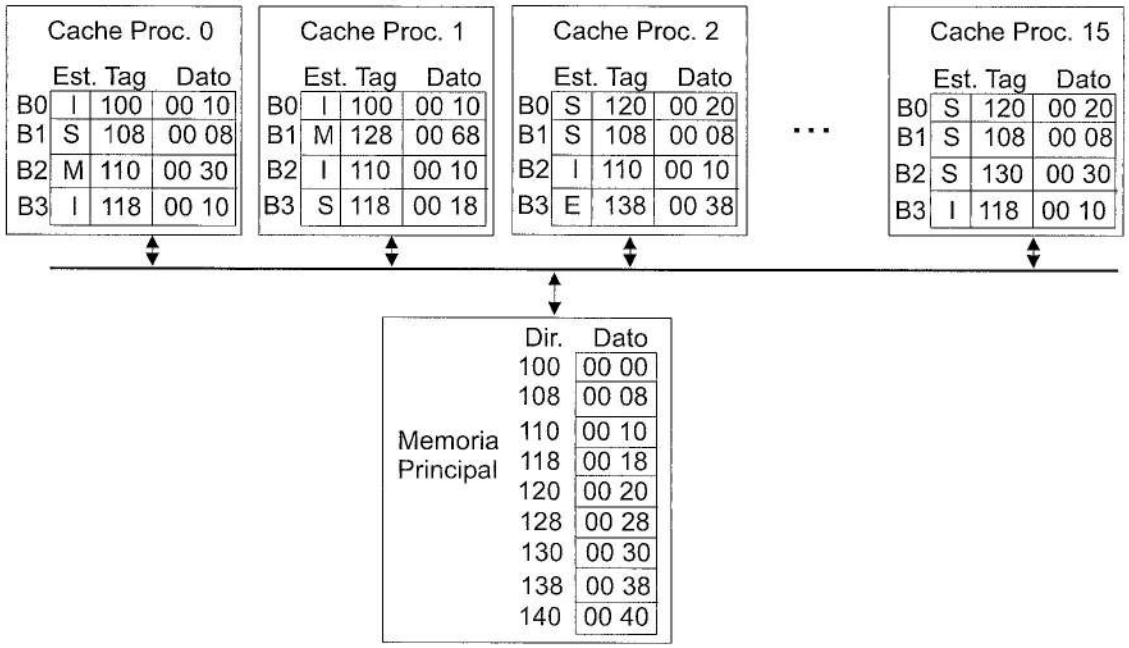


Figura 1 Multiprocesador con coherencia de caches basada en MESI

a) Procesador 0 lee el dato de la dirección 120.

b) Procesador 1 lee el dato de la dirección 100.

c) Procesador 0 escribe 80 en la dirección 110.

d) Procesador 0 escribe 80 en la dirección 120.

e) Procesador 15 escribe 80 en la dirección 130.

f) Procesador 2 lee el dato de la dirección 110.

g) Procesador 2 escribe 80 en la dirección 138.

h) Procesador 1 escribe 80 en la dirección 130.

i) Procesador 0 lee el dato de la dirección 138.

j) Procesador 0 escribe 80 en la dirección 130.

SOLUCIÓN

a) Procesador 0 lee el dato de la dirección 120. Genera una petición de lectura en el bus, dirección 120. El contenido de la cache será P0.B0 (S 120 00 20)

b) Procesador 1 lee el dato de la dirección 100. Genera una petición de lectura en el bus, dirección 100. Si la línea de *shared* no se activa tendrá P1.B0 (E 100 00 00) y si se activa será P1.B0 (S 100 00 00)

c) Procesador 0 escribe 80 en la dirección 110. No genera tráfico en el bus. El contenido de la cache será P0.B2 (M 110 00 80)

d) Procesador 0 escribe 80 en la dirección 120. Genera una petición en el bus de lectura con invalidación, dirección 120. Al final el contenido de las caches cambia en: P0.B0 (M 120 00 80) P2.B0 (I 120 00 20) P15.B0 (I 120 00 20)

e) Procesador 15 escribe 80 en la dirección 130. Genera una petición de invalidación sobre la dirección 130. El contenido de la cache será P15.B2 (M 130 00 80)

f) Procesador 2 lee el dato de la dirección 110. Genera una petición de lectura en el bus, dirección 110. Al final el contenido de las caches cambia en: P0.B2 (S 110 00 30) P2.B2 (S 110 00 30) y la memoria M[110] = 00 30

g) Procesador 2 escribe 80 en la dirección 138. No genera tráfico en el bus.El contenido de la cache será P2.B3 (M 138 00 80)

h) Procesador 1 escribe 80 en la dirección 130. Genera una petición de lectura con invalidación sobre la dirección 130. Al final el contenido de las caches cambia en: P1.B2 (M 130 00 80) P15.B2 (I 130 00 30)

i) Procesador 0 lee el dato de la dirección 138. Genera una petición de lectura en el bus, dirección 138. Al final el contenido de las caches cambia en: P0.B3 (S 138 00 38) P2.B3 (S 138 00 38)

j) Procesador 0 escribe 80 en la dirección 130. Genera una petición en el bus de lectura con invalidación, dirección 130. Al final el contenido de las caches cambia en: P0.B2 (M 130 00 80) P15.B2 (I 130 00 30) y la memoria M[110] = 00 30

1 ***(1 punto)*** *Detalle en qué sobrecoste se incurre durante una operación de E/S al ser gestionada mediante interrupciones en lugar de mediante E/S programada.*

Justifique por qué se usa dicha técnica a pesar del sobrecoste en que se incurre.

SOLUCIÓN

Secuencia de reconocimiento de interrupción (SRI): Secuencia de microoperaciones que permite saltar a la rutina de tratamiento de interrupción (RTI) salvando el estado del programa interrumpido (PC, RE, BMI).

Salvar/Recuperar registros: Salvaguarda/Recuperación del estado de ejecución del programa interrumpido que es modificado por la RTI.

Paso de parámetros: Lectura y Escritura de las variables globales que contienen los parámetros de la operación de E/S.

Retorno de interrupción (RETI): Recuperación del estado modificado por la SRI para reanudar la ejecución del programa interrumpido.

En muchos casos, este sobrecoste es muy inferior al beneficio obtenido al ceder al módulo de E/S la labor de sincronización.

2 ***(1 punto)*** *Describa brevemente las técnicas de identificación del módulo de E/S que solicita una interrupción.*

SOLUCIÓN

Identificación por muestreo o polling: Una única rutina de tratamiento de interrupción lee sucesivamente los registros de estado de los módulos de E/S para identificar cuál de ellos solicita la interrupción.

Identificación por vectorización: Durante la secuencia de reconocimiento de interrupción, la CPU inicia un ciclo de reconocimiento de interrupción, identificado por la activación de la señal INTA, para solicitar al módulo de E/S que solicita la interrupción que coloque en el bus de datos su identificación (vector de interrupción).

3 *Sea un computador con un procesador de 32 bits con una capacidad de procesamiento de 1.000 MIPS en el que la secuencia de reconocimiento de interrupciones tiene una duración de 6ñs. A este computador están conectados los siguientes periféricos:*

- *Red Ethernet.*
  - *Velocidad de transmisión de 1 Gigabit por segundo (10^9 bits/s).*
  - *Bloques de longitud variable entre 64 y 1.536 bytes.*
  - *2 registros de datos de 32 bits.*
  - *La rutina de inicio de una operación de E/S tiene 100 instrucciones.*
  - *La rutina de tratamiento de las interrupciones para transferir los datos al módulo de E/S consta de 24 instrucciones. En las operaciones de salida (transmisión), el módulo de E/S solicita una primera interrupción para cargar sus registros de datos y, a continuación, inicia la transmisión del bloque de datos.*
  - *Una vez finalizada, el módulo de E/S solicita una interrupción adicional para señalar el fin de la transmisión del bloque de datos. La rutina de tratamiento de esta interrupción consiste en la ejecución de 200 instrucciones para dar fin a la operación de E/S.*
- *Disco duro.*
  - *Velocidad de transferencia: 10^8 bytes/s.*
  - *Tiempo medio de acceso: 4 ms.*
  - *Tamaño del sector: 512 bytes.*
  - *1 Registro de datos de 32 bits.*
  - *Las rutinas de inicio y fin de una operación de E/S tienen 80 y 400 instrucciones respectivamente.*
  - *Operación por acceso directo a memoria (DMA).*

En este computador el protocolo de concesión y liberación de los buses dura un total de 2ñs y un ciclo de acceso a memoria dura 1ñs.

a) (1 punto) *Calcule cuántos módulos de red Ethernet podrían operar simultáneamente.*

b) (1 punto) *Calcule cuántos módulos de disco duro podrían operar simultáneamente, si además se encuentra operando un módulo de red Ethernet.*

c) (2 puntos) *Calcule el tiempo total de una operación en la red Ethernet para transmitir un bloque de 1,5 kB así como el porcentaje de tiempo que deja libre a la CPU.*

d) (2 puntos) *Calcule el tiempo total de una operación de lectura de un sector del disco duro así como el porcentaje de tiempo que deja libre a la CPU.*

A partir del instante t=0, comienza la lectura de un fichero de 15 kB de la unidad de disco y, a medida que van estando disponibles sus datos, se transmiten simultáneamente por la red Ethernet en bloques de 1,5 kB.

e) (1 punto) *Calcule el instante t_{fin} en el que termina la transmisión del último bloque del fichero.*

f) (1 punto) *Calcule el consumo de CPU ocasionado por la lectura y transmisión del fichero.*

SOLUCIÓN

a) Se calcula la capacidad de procesamiento necesaria para atender las interrupciones de transmisión de datos de la red.

$$\begin{aligned} CP_{Ethernet} &= Freq_{int} \times I_{int} = \frac{10^9\ bits/s}{2 \times 32\ bits} \times (6 \cdot 10^{-9}\ s \cdot 10^9\ instr/s + 24\ instr) = \\ &= \frac{10^9\ bits/s}{64\ bits} \times 30\ instr = 468,75 \cdot 10^6\ instr/s = 468,75\ MIPS \end{aligned}$$

Se podrían atender a un máximo de 1.000 MIPS/468,75 MIPS = 2 unidades de red.

b) Se calcula la capacidad de procesamiento necesaria para atender los robos de ciclo del disco duro. Como tiene 1 registro de datos de 32 bits, trabajará mediante robo de ciclo aislado.

$$t_{robo-ciclo} = 2\ ns + 1 \times 1\ ns = 3\ ns$$

$$CP_{HD} = Freq_{robo-ciclo} \times I_{robo-ciclo} = \frac{10^8\ bytes/s}{4\ bytes} \times (3 \cdot 10^{-9}\ s \cdot 10^9\ instr/s) =$$
$$= \frac{10^8\ bytes/s}{4\ bytes} \times 3\ instr = 75 \cdot 10^6\ instr/s = 75\ MIPS$$

Se podrían atender a un máximo de (1.000 MIPS-468;75 MIPS)/75 MIPS = 7 unidades de disco.

c)

$$t_{op-Ethernet} = t_{ini} + t_{primera-int} + t_{trans} + t_{última-int} = t_{ini} + (t_{sri} + t_{rti}) + t_{trans} + (t_{sri} + t_{fin}) =$$
$$= \frac{100\ instr}{10^9\ instr/s} + (6\ ns + \frac{24\ instr}{10^9\ instr/s}) + \frac{1.536\ bytes \times 8\ bits/bytes}{10^9\ bits/s} + (6\ ns + \frac{200\ instr}{10^9\ instr/s}) =$$
$$= 100\ ns + 6\ ns + 24\ ns + 12.288\ ns + 6\ ns + 200\ ns = 12.624\ ns$$

$$t_{cpu-Ethernet} = t_{ini} + N_{int} \times t_{int} + t_{última-int} = t_{ini} + N_{int} \times (t_{sri} + t_{rti}) + (t_{sri} + t_{fin}) =$$
$$= 100\ ns + \frac{1.536\ bytes}{2 \times 4\ bytes} \times (6\ ns + 24\ ns) + (6\ ns + 200\ ns) =$$
$$= 100\ ns + 5.760\ ns + 206\ ns = 6.066\ ns$$

$$\%_{cpu-libre-Ethernet} = \frac{t_{op-Ethernet} - t_{cpu-Ethernet}}{t_{op-Ethernet}} \times 100 = \frac{12.624\ ns - 6.066\ ns}{12.624\ ns} \times 100 = 51,95\,\%$$

d)

$$t_{op-HD} = t_{ini} + t_{acc} + t_{trans} + t_{último-robo-ciclo} + t_{int} =$$
$$= t_{ini} + t_{acc} + t_{trans} + t_{último-robo-ciclo} + (t_{sri} + t_{fin}) =$$
$$= \frac{80\ instr}{10^9\ instr/s} + 4\ ms + \frac{512\ bytes}{10^8\ bytes/s} + 3\ ns + (6\ ns + \frac{400\ instr}{10^9\ instr/s}) =$$
$$= 80\ ns + 4 \cdot 10^6\ ns + 5.120\ ns + 3\ ns + (2\ ns + 400\ ns) = 4.005.609\ ns$$

$$t_{cpu-HD} = t_{ini} + N_{robo-ciclo} \times t_{robo-ciclo} + t_{int} =$$
$$= 80\ ns + \frac{512\ bytes}{1 \times 4\ bytes} \times 3\ ns + (6\ ns + 400\ ns) = 80\ ns + 384\ ns + 406\ ns = 870\ ns$$

$$\%_{cpu-libre-HD} = \frac{t_{op-HD} - t_{cpu-HD}}{t_{op-HD}} \times 100 = \frac{4.005.609\ ns - 870\ ns}{4.005.609\ ns} \times 100 = 99,98\,\%$$

e) Para leer del disco y transmitir por la Ethernet en bloques de 1,5 kB un fichero de 15 kB, se leen sus 30 sectores del disco y se transmiten 10 bloques por la red Ethernet. Sin embargo, hay que tener en cuenta que cada vez que se leen 3 sectores del disco se procede a transmitir el correspondiente bloque de 1,5 kB por la red Ethernet.

A partir de los resultados de apartados anteriores se comprueba que una operación en la red Ethernet es más corta que 3 operaciones en el disco. Por lo tanto, la red está siempre disponible para transmitir el siguiente bloque y así las operaciones en el disco duro se realizan consecutivamente.

De modo que la transmisión del último bloque del fichero comienza cuando finaliza la trigésima operación del disco duro y el instante t_{fin} en el que termina será:

$$t_{fin} = 30 \times t_{op-HD} + t_{op-Ethernet} = 30 \times 4.005.609\ ns + 12.624\ ns = 120.180.894\ ns = 120.180,894\ \mu s$$

f) El tiempo total empleado por la CPU es el empleado en 30 operaciones del disco y 10 operaciones de la red Ethernet.

$$t_{total-cpu} = 30 \times t_{cpu-HD} + 10 \times t_{cpu-Ethernet} = 30 \times 870\ ns + 10 \times 6.066\ ns = 86.760\ ns = 86,760\ \mu s$$

1 *[8,5 puntos] Sea un procesador con tamaño de palabra de 32 bits y direccionamiento a nivel de byte, cuyo sistema de memoria tiene las siguientes características:*

- *Memoria principal con tiempo de acceso de 50ns.*
- *Memorias caché de nivel 1 separadas para instrucciones y datos. Las características de la caché de datos (McaD) son las siguientes:*
  - *Capacidad 32KB, bloques de 16B y tiempo de acceso 1ns.*
  - *Ubicación directa y política de lectura out of order fetch.*
  - *Política de escritura aplazada con actualización (CBWA). En los fallos de escritura, se escribe primero en Mp y posteriormente se lleva el bloque actualizado a la McaD.*
  - *El tiempo empleado en la transferencia de un bloque entre Mp y McaD es 80ns.*

En este procesador se va a ejecutar un programa, del que se ha extraido un fragmento de código para analizar el rendimiento de la McaD:

```
for (i=0; i<64; i=i+1)
   for (j=0; j<64; j=j+2)
      C[i][j] = B[i][j] + A[i][j];
```

Sabiendo que:

- *Las variables i y j están asignadas a registros del procesador y cada una de las matrices utilizadas consta de 4.096 elementos de 1 palabra y están almacenadas en memoria por filas.*
- *La dirección de Mp a partir de la que está almacenada la matriz A es H´08000, y las matrices B y C están almacenadas, en este orden, a continuación de A.*
- *Al iniciarse la ejecución de este bucle no se encuentra en la McaD ningún elemento de ninguna de las matrices.*

a) *Calcule el número de bloques que ocupan las matrices y cuántos elementos se almacenan en cada bloque.*

b) *Especifique cómo interpreta la McaD las direcciones físicas y en qué bloques de la McaD se alojarán las diferentes matrices.*

c) *Rellene la siguiente tabla, para las cuatro primeras iteraciones (observe que la variable j, que controla el bucle más interno, se incrementa en 2 unidades en cada iteración). Calcule a continuación:*

- *El número de aciertos y fallos que se producen en la McaD en el acceso a las matrices, para la ejecución completa del código, distinguiendo entre lectura y escritura, así como la tasa de aciertos.*
- *La probabilidad de reemplazar un bloque modificado.*
- *El tiempo medio de ocupación empleado en los accesos a las matrices.*

Traza	*A(0,0)*	*B(0,0)*	*C(0,0)*									
Acierto/Fallo												
Lect./Esc.												
BloqueMcaD												
Reemplazo de												

d) *Este mismo fragmento de código se ejecuta ahora en otro procesador cuya McaD difiere únicamente de la del procesador anterior en la política de ubicación, que en este caso es asociativa por conjuntos. Indique razonadamente cuántos bloques debería tener como mínimo cada conjunto para poder obtener la tasa de aciertos máxima y qué valor tendría dicha tasa.*

e) *Suponga finalmente que el procesador tiene memoria virtual paginada con las siguientes características:*

- *Páginas de 4KB*
- *3 niveles de tablas de páginas y una TLB asociativa con 32 entradas y tiempo de acceso de 1ns*

e.1) *Calcule la tasa de aciertos de la TLB en el acceso a las matrices suponiendo que inicialmente está invalidada.*

e.2) *Calcule los tiempos máximo y mínimo de ocupación en el acceso a datos suponiendo que no se produce fallo de página y que todas las tablas de traducción se encuentran en Mp. Utilice para realizar estos cálculos los datos del procesador original (McaD directa).*

SOLUCIÓN

a) Como cada matriz consta de 4.096 elementos de 4B y los bloques son de 16B (4 elementos por bloque), cada una ocupa 1.024 bloques ((4.096x4)/16). En el código se referencian 3 matrices de las mismas dimensiones, que en total ocupan 3.072 bloques.

b) La McaD tiene una capacidad de 32KB, por lo que está compuesta por: $32KB/(16B/bloque) = 2^{11} = 2.048$ *bloques*. Al ser la política de ubicación directa, la McaD interpreta las direcciones físicas, suponiendo que estas son de 32 bits, de la siguiente forma:

etiqueta	bloque-Mca	byte en bloque
17 bits	11 bits	4 bits

Como la matriz A comienza en la dirección de Mp H'08000, dicha dirección es interpretada por la McaD de la forma:

000 ... 001	000 0000 0000	0000
etiqueta	bloque-Mca	byte

El primer bloque de A se alojará en el bloque 0 de la caché, el segundo en el bloque 1, y así sucesivamente hasta el último bloque, que se alojará en el bloque 1.023 de la caché. Puesto que la matriz B está almacenada en Mp justo a continuación de A, y tiene el mismo tamaño, se alojará en los bloques 1.024 a 2.047 de la caché. Finalmente la matriz C, almacenada en Mp a continuación de B, se alojará de nuevo en los bloques 0 a 1.024 de la caché.

c) El comportamiento en las cuatro primeras iteraciones, teniendo en cuenta que, según lo expuesto en el apartado anterior A y C se alojan en los mismos bloques de caché es el siguiente:

Traza	A(0,0)	B(0,0)	C(0,0)	A(0,2)	B(0,2)	C(0,2)	A(0,4)	B(0,4)	C(0,4)	A(0,6)	B(0,6)	C(0,6)
Acierto/Fallo	F	F	F	F	A	F	F	F	F	F	A	F
Lect./Esc.	L	L	E	L	L	E	L	L	E	L	L	E
BloqueMcaD	0	1024	0	0	1024	0	1	1025	1	1	1025	1
Reemplazo de	-	-	A(0,0)	C(0,0)	-	A(0,2)	-	-	A(0,4)	C(0,4)	-	A(0,6)

- En el acceso a las matrices A y C todas las referencias producen fallo, por lo que se dan 2.048 fallos de lectura y 2.048 fallos de escritura. Obsérvese que al realizarse el bucle más interno 32 veces se referencian únicamente la mitad de los elementos de las matrices. En el acceso a B una de cada 2 referencias produce fallo, por lo que se dan 1.024 aciertos y 1.024 fallos de lectura.

  El número total de accesos a la McaD es $64 \times 32 \times 3$ *accesos* $= 6.144$ *accesos*, y por lo tanto:

  $Hr_{McaD} = 1.024/6.144 = 0,1667 = 16,67\,\%$

- La probabilidad de reemplazar un bloque modificado es 0 ya que cuando se produce un fallo en escritura primero se escribe en Mp y a continuación se lleva el bloque ya actualizado a McaD. Como los únicos bloques a los que se accede en escritura son los de C, que son desalojados por los de A antes de ser modificados en caché, nunca se reemplazan bloques modificados.

- El tiempo medio de ocupación teniendo en cuenta los resultados anteriores es el siguiente:

$$\frac{1.024\ aciertos\times 1ns\ +\ (2.048+1.024)\ FL\times(1+80)ns\ +\ 2.048\ FE\times(1+50+80)ns}{6.144\ accesos} = 84,33\text{ns}$$

Obsérvese que el tiempo empleado en los fallos de lectura (*FL*) y en los de escritura (*FE*) es distinto ya que en los fallos de escritura primero se escribe en Mp (50ns) y a continuación se lleva el bloque actualizado a McaD (80ns), mientras que en los fallos de lectura se envía el bloque a la McaD comenzando por la palabra en la que se produce el fallo (*out of order fetch*), que se envía además a la CPU.

d) Si cada conjunto tuviera 2 bloques, el número de conjuntos de la McaD sería 2^{11} *bloques*$/(2$*bloques*$/$*conjunto*$) =$ 1.024 *conjuntos*, alojándose las tres matrices en los mismos conjuntos (0 a 1.023), por lo que se producirían fallos por conflicto, adicionales a los forzosos.

Por lo tanto, cada conjunto debería tener como mínimo 4 bloques. De este modo la McaD estaría formada por 2^{11} *bloques*$/(4$*bloques*$/$*conjunto*$) = 512$ *conjuntos*, pudiendo residir simultáneamente en caché los 3 bloques

de A, B y C que se necesitan en cada momento, por lo que únicamente se producirían fallos forzosos, y en consecuencia se conseguiría la tasa mínima de fallos. Ya que de cada bloque se utilizan únicamente dos elementos, el primero produciría fallo y el otro acierto, por lo que dicha tasa sería del 50 %.

e) Suponiendo que las direcciones virtuales de comienzo de las matrices están alineadas a página, cada matriz ocupa $(4.096 \times 4)B/(4KB/pagina) = 4$ páginas, ocupando en total 12 páginas.

e.1) Ya que la TLB es asociativa y dispone de 32 entradas, se producirían únicamente 12 fallos correspondientes a la primera vez que se hace referencia a cada una de las diferentes páginas, siendo la tasa de aciertos de la TLB:

$$Hr_{TLB} = (6.144 - 12)/6.144 = 0,998 = 99,8\,\%$$

e.2) En primer lugar, comprobaremos si se puede solapar el acceso a la TLB y a la caché directa. Para ello hay que comprobar si con la parte de la Dv que no se traduce (los 12 bits menos significativos, ya que las páginas son de 4KB) se pueden direccionar todos los bloques y los bytes dentro del bloque de la McaD. En este caso, esto no sería posible ya que se necesitan 11 bits para seleccionar el bloque más 4 para el byte dentro del bloque.

Teniendo en cuenta lo anterior, los tiempos mínimo y máximo de ocupación serían los siguientes:

$$T_{min} = T_{TLB} + T_{McaD} = 1 + 1 = 2\ ns$$

Este tiempo se obtiene cuando hay acierto tanto en TLB como en caché.

$$T_{max} = T_{TLB} + T_{TP's} + T_{Mca} + T_{Mp} + T_{blq} = 1 + 3 \times 50 + 1 + 50 + 80 = 282\ ns$$

Este tiempo se obtiene cuando hay fallo en la TLB, por lo que se deben consultar los 3 niveles de tablas residentes en Mp para realizar la traducción, y se produce un fallo de escritura en la caché, por lo que primero se escribe en Mp y a continuación se envía el bloque, ya actualizado, a la caché.

NOTA.- Según se vió en el apartado c), no hay que considerar el tiempo empleado en reemplazar un bloque modificado ya que no se puede dar esta circunstancia.